P
E
E
R

A Brief Introduction to C++

Frank McKenna

OpenSees Developers Workshop
Berkeley, CA
August 25, 2005

Sponsored by the National Science Foundation
through the Pacific Earthquake Engineering Research Center
and through NEESit

C Basics

- Fundamental data types: char, int, float, double
- Derived Types: pointers, arrays, structures
- Variables
- Operators =, +, -, *, /, %, <, <=, >, >=, ==, !=
- Control-flow Constructs: if-else, while, do, for
- Procedures
- Libraries, lots of libraries.
- C Help:
  - The C Programming Language, Brian W. Kernighan and Dennis W. Ritchie, Prentice-Hall.

Hello World!

```
#include <stdio.h>

main() {
  printf(“Hello World!”);
}
```

Variables and Pointers

- A variable in a program is something with a name, the value of which can vary. In a running program, the compiler/linker assigns a specific block of memory to hold the value of the variable.

```
int k;     /* declaration of variable k to be an int */
k = 2;     /* set the value of k to be 2    */
```

- A pointer variable is a variable designed to hold an address of a block of memory.

```
int *kPtr;      /* declaration of variable kPtr to hold the address of an int*/
kPtr = &k;   /* set the value of kPtr to be address of k */
*kPtr  = 2;   /* set the value of what kPtr is pointing to to be 2*/
```

Arrays and Structures

- An array is a contiguous block of memory.☐

```
int kArray[10];   /* declaration of variable kArray to be an array of 10 integers */
kArray[0] = 2;     /* set the value of the first to be 2 */
kPtr = &kArray[9];  /* set the value of kPtr to be address of last element of array*/
```

- A structure is a user defined collection of data. Unlike arrays, where all members have same data type, structures can group together variables of different data types.

☐

```
typedef struct truss {
    int tag;
    int nodes[2];
    int node2;
    double A;
    double E;
 } Truss;
```

```
Truss t1;   /* struct truss t1
Truss *elePtr = &t1;

t1.nodes[0] = 2;
(*elePtr).nodes[1]=3;
```

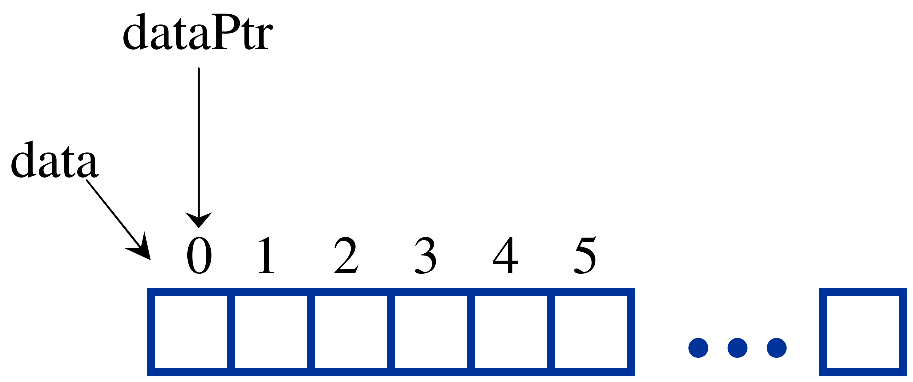
Example – don’t do this at home!

```
#include <stdio.h>
#include <stdlib.h>
#define size 10
double sumValues(int, double *);
main () {
   int i;
   double values[size];
   for (i=0; i<size; i++) {
       values[i] = rand();
        printf(“random number: %f\n”,values[i];
   }
   sum = sumValues(size, values);
   printf(“\n sum of numbers: %f\n”,sum);
}
```

```
double sumValues(int n, double *data) {
   int i =0;
   double sum =0.0;
   while (i < n) {
       sum = sum+data(i);  /* sum +=data(i) */
       i = i+1;               /* i++ */
   }
   return sum;
}
```

sumValues() - with a pointer and a do

```
double sumValues(int n, double *data) {
    int i =0;
    double sum =0.0;
    double *dataPtr = data;
    do {
      sum += *dataPtr;
      i++;
      dataPtr++;
  } while (i < n)
  return sum;
}
```



C++ Basics

- C++ is an extension of the C language
  - adds REFERENCES
  - adds CLASSES

- C++ Help:
  - An Introduction to Object-Oriented Design in C++, Jo Ellen Perry & Harold D. Levin, Addison-Wesley.
  - C++ How to Program, H.M. Deitel and P.J.Deitel, Prentice-Hall.

C++ Pointers and References

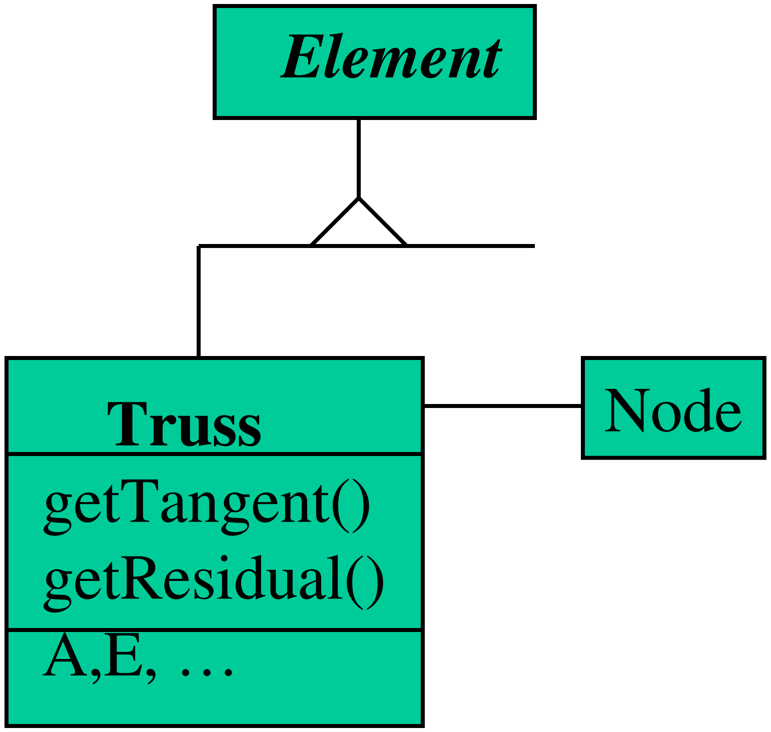
```
void sum(double a, double b, double *c) {
    double result = a + b;
    *c = result;
}
```

```
void sum(double a, double b, double &c) {
    double result = a + b;
    c = result;
}
```

C++ Classes

- A class is a C++ construct to hold both data and functions in the same block of memory.
- Classes typically have a definition which outlines the functions and variables, and their accessibility (public, protected, private). The definition is typically placed in a header file.
- Class also has an implementation. This is where the functions (methods) are defined. This is (typically) placed in a separate file, the implementation file.
- A Class can inherit both variables and implementation from a parent class. This is termed **inheritance**.
- A Class can override (redefine) the methods of the parent class. This is termed **polymorphism**.

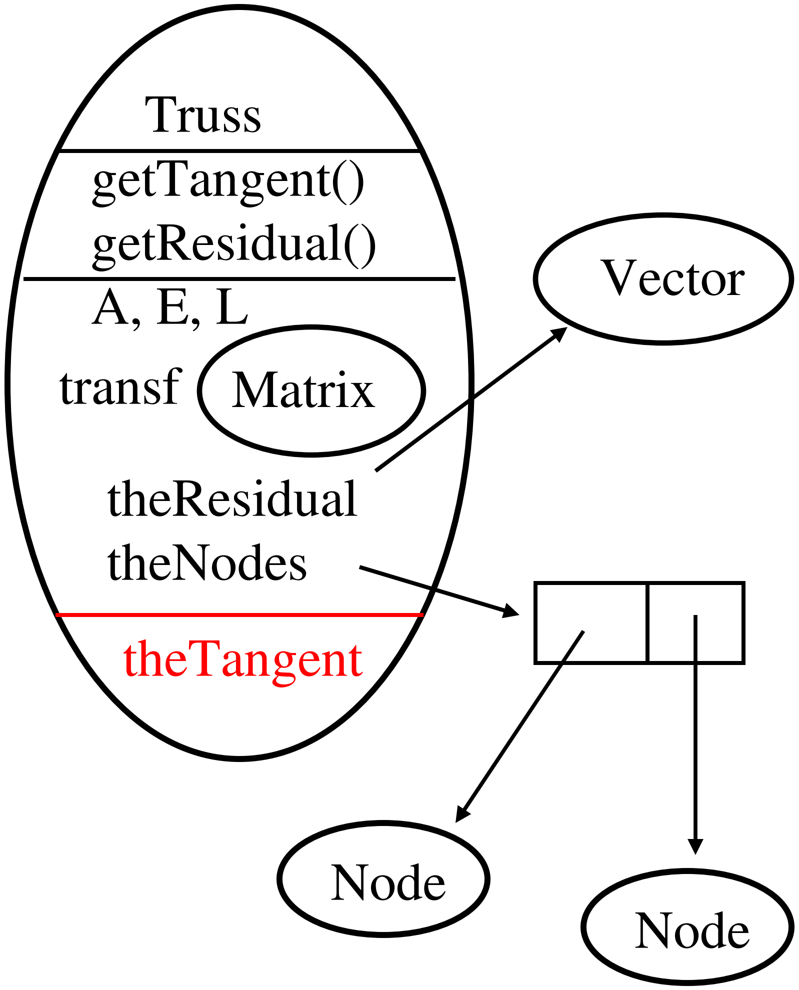
Simple Truss Example



```
class Truss : public Element {
    public:
        Truss(double A, double E,
            Node *node1, Node *node2);
        ~Truss();
        const  Matrix &getTangent();
        const  Vector &getResidual();
    private:
        double computeTrialStrain();
        double A, E, L;
        Matrix transf;
        Vector *theResidual;
        Node **theNodes;
        static Matrix theTangent;
};
```

Constructor

```
Truss::Truss(double a, double e,
                 Node *node1, Node *node2)
:Element(), A(a), E(e), transf(1,4)
{           theResidual = new Vector(4);
            theNodes = new Node*[2];
            theNodes[0] = node1;
            theNodes[1] = node2;
            Vector &crd1 = node1->getCrds();
            Vector &crd2=  node2->getCrds();
            double dx = crd2(0) - crd1(0);
            double dy = crd2(1) – crd1(1);
            L = sqrt(dx * dx + dy * dy);
            double cs = dx/L; double sn = dy/L;
            trans(0,0) = -cs;  trans(0,1) = -sn;
             trans(0,2) =  cs; trans(0,3) =  sn;
}
```

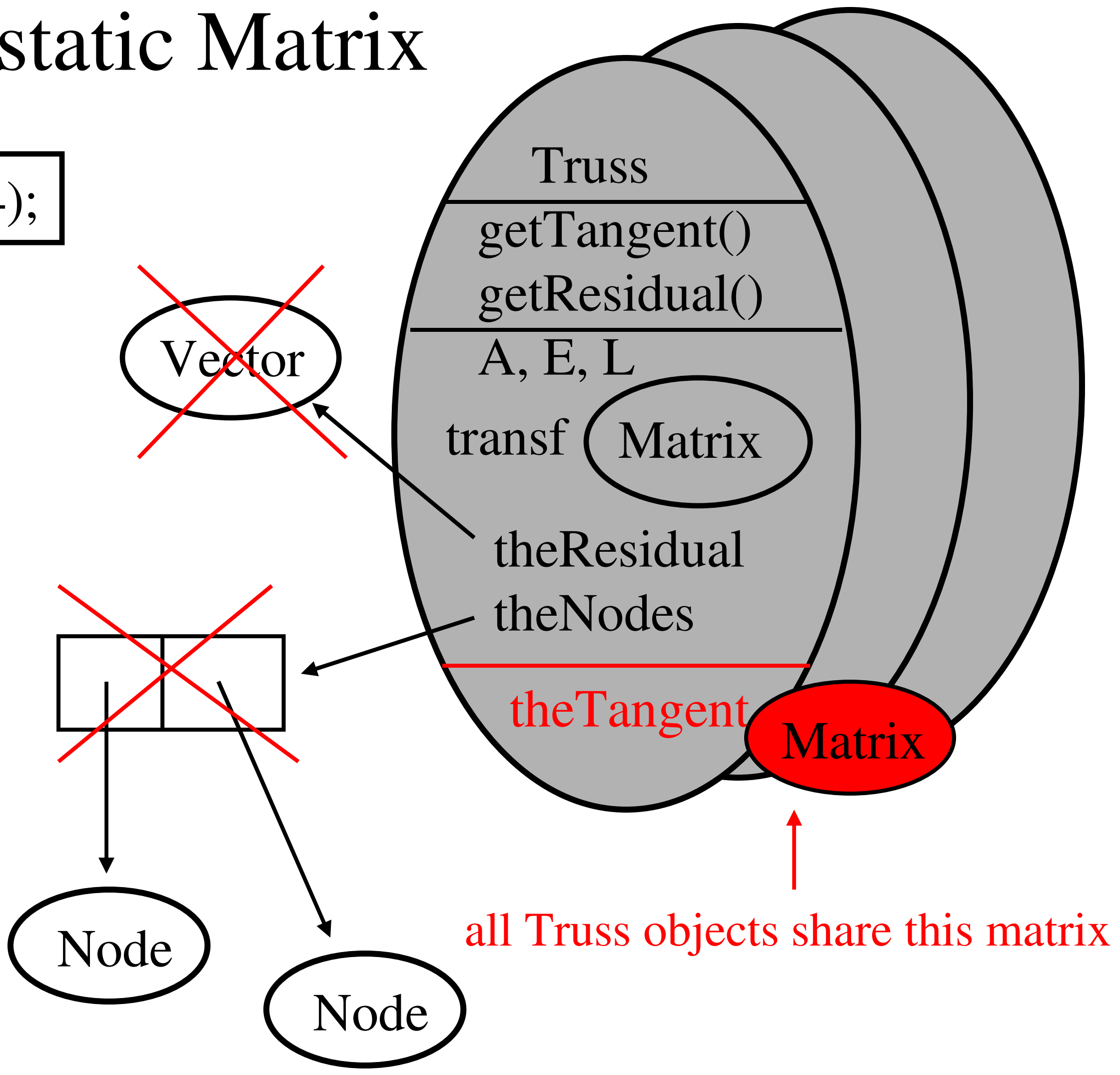


And remember that static Matrix

```
Matrix Truss::theTangent(4,4);
```

Destructor

```
Truss::~Truss()
{
    delete theResidual;
    delete [ ] theNodes;
}
```



Typically only delete objects you constructed

Public Methods

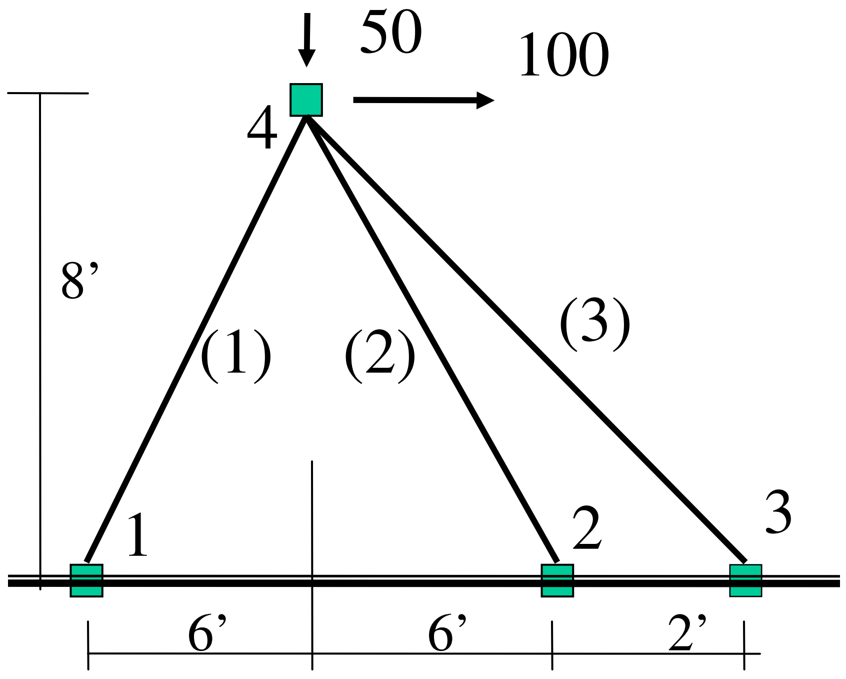
```
const Matrix &getTangent(void) {
        theMatrix = transf ^ transf;
        theMatrix *= A * E / L;
        return theMatrix;
}

const Vector &getResidual() {
        double strain = this->computeStrain();
        double force = A * E / L * strain;
        Vector &resid = *theResidual;
        for (int i=0; i<4; i++)
                resid(i) = transf(0,i) * force;
        return resid;
}
```

Private Method

```
double Truss::computeTrialStrain() {
   Vector &disp1 = theNodes[0]->getTrialDisp();
   Vector &disp2 = theNodes[1]->getTrialDisp();
   double dLength = 0.0;
   for (int i=0; i<2; i++)
       dLength -= (disp2(i)-disp1(i)) * trans (0,i);
  double strain = dLength / L;
  return strain;
}
```

Remember This!

	E	A
1	3000	10
2	3000	5
3	3000	5



```
#include <Node.h>
#include <Truss.h>
#include <iostream.h>
main() {
    Node node1(   0.0,   0.0);
    Node node2(144.0,  0.0);
    Node node3(168.0,  0.0);
    Node node4(  72.0, 96.0);
    Truss truss1(10, 3000, &node1, &node4);
    Truss truss2( 5, 3000, &node2, &node4);
    Truss truss3(5, 3000, &node3, &node4);

    opserr << truss1.getTangent();


            .. NEED OpenSees CODE
              to do anything useful!
}
```

BREAK